CSE 3302
Programming Languages

Smalltalk

Chengkai Li
Fall 2007

Readings:

- Smalltalk by Example, Chapter 1-6
- (Chapter 7-9 are also recommended, especially sections related to Array)

The language is together with its interactive runtime system

Smalltalk Runtime

- Runtime written in the language itself
- Can change the system on-the-fly
- Debug system state (image), object, class hierarchy

Squeak Demo

- The files (.exe .changes, .img, .sources)
- Menu and Mouse
  (red-button (usually left button), yellow-button (usually right button), blue-button(Alt+red, "morphic halo"))
- Save changes into .img and .sources
- Transcript: system console, log
- Workspace: Run code snippet, text documents, …
- do it, print it, inspect it, explore it
- System Browser (Class, Object, Instance variable, Method)
- Hierarchy Browser
- Method Finder

Demo: Race Car

Based on
http://www.youtube.com/watch?v=y_3l08tI5wQ

Demo: Quinto Game

Squeak by Example, Chapter 2

Demo: Quinto Game

1. Create new class category: `SBE-Quinto`
2. Define new class: `SBECell`

```
Object subclass: #NameOfSubclass
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'SBE-Quinto'
```

```
SimpleSwitchMorph subclass: #SBECell
instanceVariableNames: 'mouseAction'
classVariableNames: ''
poolDictionaries: ''
category: 'SBE-Quinto'
```

Message: class compiled

Demo: Quinto Game

3. Add methods: `initialize`

```
initialize
  super initialize.
  self label: ''.
  self borderWidth: 2.
  bounds := 0@0 corner: 16@16.
  offColor := Color paleYellow.
  onColor := Color paleBlue darker.
  self useSquareCorners.
  self turnOff
```

```
newCell := SBECell new.

Message initialize will be automatically sent to the newly created object when the class has an initialize method.

newCell initialize.
```


Demo: Quinto Game

4. Inspect an object

```
SBECell new  (inspect it)
self openInWorld (do it)
```

5. Add new class: SBEGame

```
BorderedMorph subclass: #SBEGame
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'SBE--Quinto'
```


Demo: Quinto Game

6. Add methods: `SBEGame>>initialize`

```
initialize
  | sampleCell width height n |
  super initialize.
  n := self cellsPerSide.
  sampleCell := SBECell new.
  width := sampleCell width.
  height := sampleCell height.
  self bounds: (5@5 extent: ((width*n) @(height*n)) + (2 *
  self borderWidth)).
  cells := Matrix new: n tabulate: [ :i :j | self
  newCellAt: i at: j ].
```

7. Put methods into category (protocol)

Demo: Quinto Game

8. Two more methods for SBEGame

```
SBEGame»cellsPerSide
  "The number of cells along each side of the game"
  ^10
```

For annotation only,

not the language syntax.

```
SBEGame»toggleNeighboursOfCellAt: i at: j
   (i > 1) ifTrue: [ (cells at: i -- 1 at: j ) toggleState].
   (i < self cellsPerSide) ifTrue: [ (cells at: i + 1 at: j) toggleState].
   (j > 1) ifTrue: [ (cells at: i at: j -- 1) toggleState].
   (j < self cellsPerSide) ifTrue: [ (cells at: i at: j + 1) toggleState].
```

Demo: Quinto Game

9. Two more methods for SBECell

```
SBECell»mouseAction: aBlock
  mouseAction := aBlock

SBECell»mouseUp: anEvent
  mouseAction value
```

10 . Try and debug

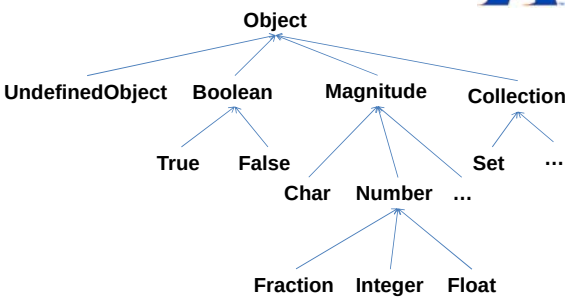
```
SBEGame>>initialize
  …
  ^cells
```


Demo: Quinto Game

- File Out and File In

Everything is object. Objects communicate by messages.

Object Hierarchy



No Data Type.
There is only Class.

Smalltalk Syntax is Simple.

Syntax

- Smalltalk is really "small"
  - Only 6 keywords (pseudo variables)
  - Class, object, variable, method names are self explanatory
  - Only syntax for calling method (messages) and defining method.
    - No syntax for control structure
    - No syntax for creating class

Expressions

- Literals
- Pseudo Variables
- Variables
- Assignments
- Blocks
- Messages

Literals

- Number: `3 3.5`
- Character: `$a`
- String: `' '` (`'Hel','lo!'` and `'Hello!'` are two objects)
- Symbol: `#` (`#foo` and `#foo` are the same object)
- Compile-time (literal) array: `#(1 $a 'abc')`
- Run-time (dynamic) array: `{1. $a. SBEGame new }`
- Comment: `"This is a comment."`

Pseudo Variables

- `true`: singleton instance of `True`
- `false`: singleton instance of `Falsee`
- `nil`: singleton instance of `UndefinedObject`
- `self`: the object itself
- `super`: the object itself (but using the selector defined for the superclass)
- `thisContext`: activation of method. (inspect the state of system)

Variables

- Instance variables.
- Local variables (method, blocks)

```
| sampleCell width height n |
```

- Arguments (method argument, block argument)

```
SBEGame»toggleNeighboursOfCellAt: i at: j
[ :i :j | self newCellAt: i at: j ]
```

- Shared Variables:
  - Global variables, e.g., `Transcript`
  - Class variables, e.g., `Epsilon` in `Float`

Conventions

- Class name, class variable, global variable:
  (Capital letter for the first character of every word)

```
Table
HashTable
```

- Local variables, arguments, instance variable:
  (Capital letter for the first character of every word, except the first word)

```
sampleCell
```

- Object (instance of a class, especially arguments)

```
aTable
aHashTable
```

Assignments

- `bounds := 0@0 corner: 16@16`

or

- `bounds _ 0@0 corner: 16@16`

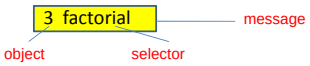
- Assignment returns value, which is the object to the left of :=.

Defining a Method

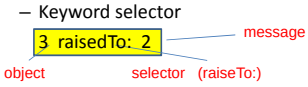
selector (method name)
| local variable |
statement (expression). (. Is used to end a statement)
statement(expression).
^ return-value (^ returns value from a method)

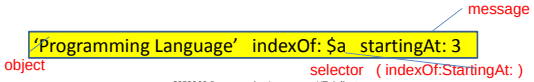
Methods and Messages

- Method Name: Selector
- Method Invocation: Message
  - Unary selector



  - Keyword selector





Keyword Selector: more readable

- `table insert: anItem at: anIndex`

`table insert: 3 at: 5`

vs.

- `table.insert(anItem, anIndex)`

`table.insert(3,5)`

Binary selector

- 2 + 3

- 2 + 3 + 4 ?

- aTable / 3 (what it means depends on the class)
- 1+2*3 (* does not have higher precedence than -, because they are messages that can be sent to any object. No mathematical meaning is assumed.)

- Examples:
  - Integer>>#+
  - Complex>>#+
  - Fraction>>#+

3/5
(1/3) + (1/2)

Binary selector

- + - * /
- = (equality) ~= >= <= > <
- == (identity, the two objects are the same unique object), ~~
- & | Boolean
- , (string concatenation)

'Hel','lo' = 'Hello'
'Hel','lo' == 'Hello'
#Hello == #Hello

- Assignment := is not a method

Expression

- Associativity for Binary selector : left to right
- Precedence rules:
  Unary selector, then Binary selector, then Keyword selector

```
1+2/4
2 raisedTo: 1 + 3 factorial
```

- ( ) for changing the order of evaluation
- "-object" was not there originally. So "3 - - 4" generated syntax errors in previous versions.

Message Cascading

- i.e., Sequence Operator

```
Transcript cr.
Transcript show: 'hello world'.
Transcript cr
```

⇒

```
Transcript cr; show: 'hello world'; cr
```


A block is an anonymous function.

Block

- Evaluate a block: `value`

The evaluation result is the object from the last statement.

```
[ 1+2 ] value
[ 1+2.'abc', 'def'] value
[ 1+2. SBEGame new] value
```


Block Parameters

- `[:x :y | x+y ] value:2 value:3`
-
```
[ :x :y |
    | z |
    z := x+ y.
    z:=z*z.
    z
] value: 2 value: 3
```


Block Closure

- Block can access variables declared in enclosing scope.

```
| x |
x := 1.
[ :y | x + y ] value: 2
[ :y | self x: + y ] value: 2
```

Block is Object!

```
z := [:x :y | x+y ].
z value:2 value:3
```


"Control Structures" by Messages

- Conditions: Messages to Boolean objects, with blocks as arguments

class `True` (subclass of Boolean, `False` is similar)

Selectors:

- `ifTrue: alternativeBlock`
  `^ alternativeBlock value`
- `ifFalse: alternativeBlock`
  `^nil`
- `ifTrue:ifFalse:`
- `ifFalse:ifTrue:`

- Example
  - `(a < b) ifTrue: [max:=b] ifFalse: [max:=a]`

"Control Structures" by Messages

- While Loops : Messages to blocks
- Example
  - n := 1.
    [n < 10] whileTrue: [n := n*2]

"Control Structures" by Messages

- Counting Loops : Blocks as both message receivers and parameters
- Example
  - `n := 1.`
    `10 timesRepeat: [ n := n*2 ]`
  - `n := 1.`
    `1 to: 10 do: [ n := n*2 ]`
  - `n := 0.`
    `1 to: 10 do: [ :i | n := n + i ]`
  - `n := 0.`
    `1 to: 10 by: 2 do: [ :i | n := n + i ]`
  - `n := 0.`
    `10 to: 1 by: -2 do: [ :i | n := n + i ]`
- Let's see how `Number>>to:do:` is implemented